ECE385

Fall 2021

Experiment 7

SOC with NIOS II in System Verilog

Ge Yuhao, Lou Haina
D231, Nov.18, 2021
TA: Chenhao Wang

1 Introduction

NIOS-II processor is an IP based 32-bit CPU which can be programmed in C. In this lab, we use NIOS-II processor as a system controller to accomplish small tasks like LED blinking and do 8-bit number reading from switches and summing them into an accumulator. In NIOS-II, we connect SDRAM controller and PIO blocks to make NIOS-II work.

2 Written Description and Diagrams of NIOS-II System

2.1 Summary of Operation

Describe in words the hardware component of the lab Because we use platform designer tool to place our hardware component, we only have platform designer module which involve IP blocks including NIOS II, SDRAM controller, PLL and some PIO blocks. NIOS II has a 32-bit modified Harvard RISC architecture with C compiler, is an IP based 32-bit CPU controller, It works as the system controller and handle tasks which do not need to be high performance. AVALON Memory Mapped Bus is a 32-bit memory mapped interface which assign a block of addresses. NIOS II uses this to interface to memory and I/O. PIO module is a bridge from AVALON to FPGA logic, PIO modules may be input(to software), output (to FPGA fabric), or bidirectional. its base address is assigned via Qsys. For external SDRAM, it needs separate PLL to phase shift external clock to make it refresh at appropriate frequency every clock. Besides, we need PLL module to do clock operations for SDRAM because SDRAm need 3ns delay then general clock. The diagram for SDRAM, SDRAM controller and PLL module are as follows:

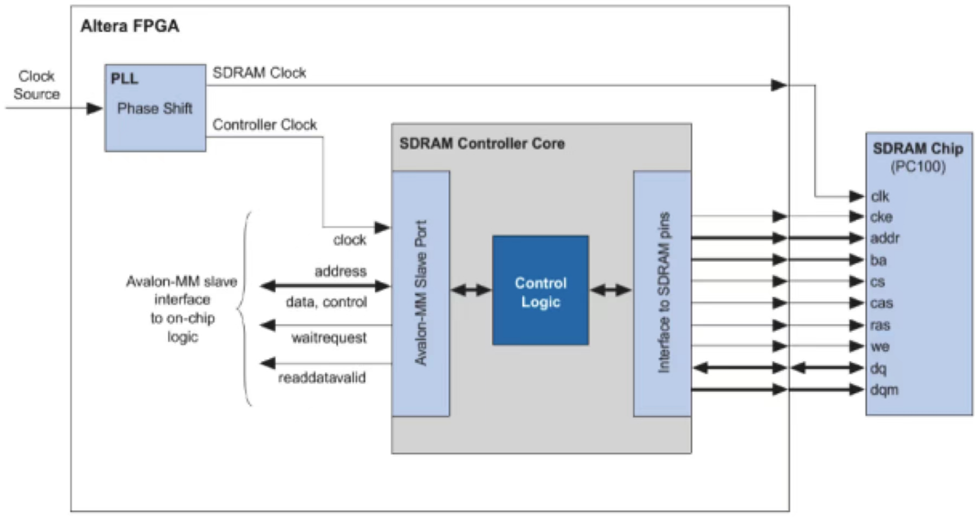


Figure 1: The Block Diagram SDRAM, SDRAM controller and PLL module

Describe in words the software component of the lab

- There are two functions our software can achieve: one is making the led blink and the other is an accumulation process.
- For our led blinking function, from code line 13 to 14, is the SET function. we set i from 0 to 100000 as delays and make the least significant bit in LED OR 1 which must output 1 therefore we make the rigt most led light up. In line 15 and 16, it's clear function, we AND least signficant bit with x0001, which is always 0, therefore we successfully clear it. And this should happen at infinite loop with set and clear happen alternatively.

- For our increment function, we use parameter halt to make function waiting for user to set switch and press run. Only when the program has halted and user press run again, it will do the next increment. The value in the switch will be added to value continuously and be displayed on LED, the value of the pointer is given by Qsys.

2.2 Top Level Block Diagram

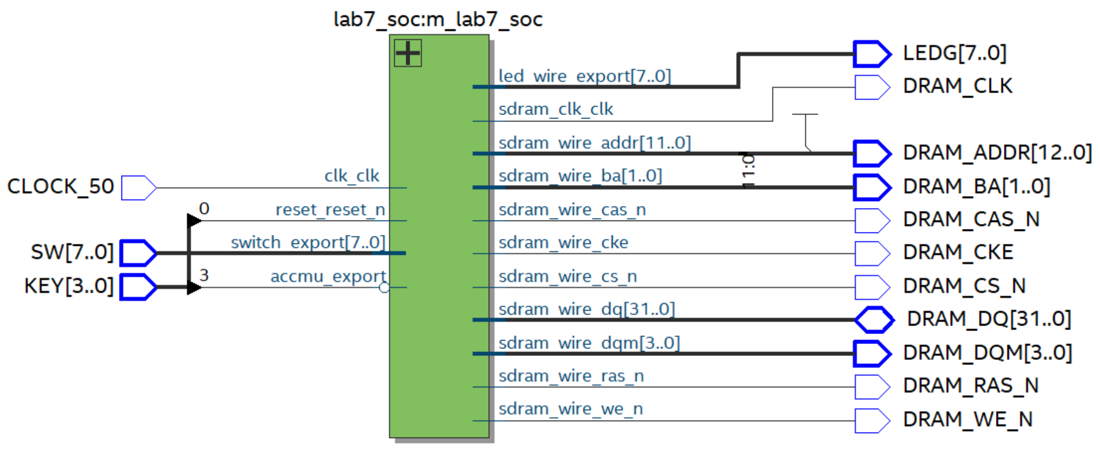


Figure 2: Top level Block Diagram

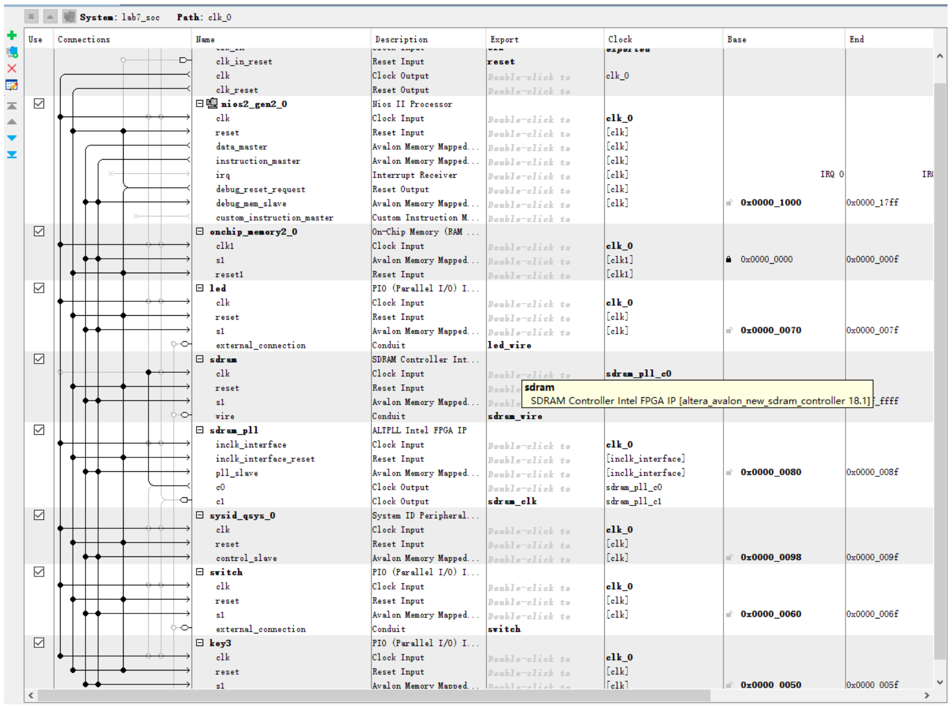
2.3 Written Description of all .sv Modules

Module: lab7.sv
Inputs: CLOCK_50, [3:0] KEY, input [7:0] SW
Outputs: [7:0] LEDG, [12:0] DRAM_ADDR,[1:0] DRAM_BA, DRAM_CAS_N, DRAM_CKE, DRAM_CS_N, [3:0] DRAM_DQM, DRAM_RAS_N, DRAM_WE_N, DRAM_CLK, [7:0] SW
inout: [31:0] DRAM_DQ
Description: This top_level module instantiates the lab7_soc module created by Qsys.
Purpose: it is the top level for integrating the Nios II system with the rest of the hardware.

Module: lab7_soc.v
Inputs: input wire accmu_export, wire clk_clk, wire reset_reset_n, wire [7:0] switch_export **inout:** wire [31:0] sdram_wire_dq
Outputs: output wire [7:0] led_wire_export, wire sdram_clk_clk, wire [11:0] sdram_wire_addr, wire [1:0] sdram_wire_ba, wire sdram_wire_cas_n, wire sdram_wire_cke, wire sdram_wire_cs_n, wire [31:0] sdram_wire_dq, wire [3:0] sdram_wire_dqm, wire sdram_wire_ras_n, wire sdram_wire_we_n **Description:** This is NIOS-II hardware system generated by platform designer.
Purpose: This module is used to connect in top-level lab7.sv to accept input and generate output

2.4 System Level Block Diagram

NIOS II has a 32-bit modified Harvard RISC architecture with C compiler, is an IP based 32-bit CPU controller, It works as the system controller and handle tasks which do not need to be high performance. In platform design, it named nios2_gen2_0, and will generate all logic operations and computations. AVALON Memory Mapped Bus is a 32-bit memory mapped interface which assign a block of addresses. NIOS II uses this to interface to memory and I/O. Onchip-memory serves as the on-chip memory in FPGA. It requires more logic elements than SDRAM and increase the speed of I/O. PIO module is a bridge from AVALON to FPGA logic, PIO modules may be input(to software), output (to FPGA fabric), or bidirectional. its base address is assigned via Qsys. LED and SWITCH and key are three PIO module, it handles LED output and its led_wire will be connected to physical led pin on FPGA board. For external SDRAM, it needs separate PLL to phase shift external clock to make it refresh at appropriate frequency every clock. It stores the software program. Besides, we need PLL module to do clock operations for SDRAM because SDRAm need 3ns delay then general clock. PLL block offers clock signal to SDRAM block.



System: lab7_soc **Path**: clk_0

Use	Name	Description	Export	Clock	Base	End
	clk_in_reset	Reset Input	**reset**			
	clk	Clock Output	Double-click to	clk_0		
	clk_reset	Reset Output	Double-click to			
✓	⊟ **nios2_gen2_0**	Nios II Processor				
	clk	Clock Input	Double-click to	**clk_0**		
	reset	Reset Input	Double-click to	[clk]		
	data_master	Avalon Memory Mapped...	Double-click to	[clk]		
	instruction_master	Avalon Memory Mapped...	Double-click to	[clk]		
	irq	Interrupt Receiver	Double-click to	[clk]	IRQ 0	IR
	debug_reset_request	Reset Output	Double-click to	[clk]		
	debug_mem_slave	Avalon Memory Mapped...	Double-click to	[clk]	**0x0000_1000**	0x0000_17ff
	custom_instruction_master	Custom Instruction M...	Double-click to			
✓	⊟ **onchip_memory2_0**	On-Chip Memory (RAM ...				
	clk1	Clock Input	Double-click to	**clk_0**		
	s1	Avalon Memory Mapped...	Double-click to	[clk1]	0x0000_0000	0x0000_000f
	reset1	Reset Input	Double-click to	[clk1]		
✓	⊟ **led**	PIO (Parallel I/O) I...				
	clk	Clock Input	Double-click to	**clk_0**		
	reset	Reset Input	Double-click to	[clk]		
	s1	Avalon Memory Mapped...	Double-click to	[clk]	**0x0000_0070**	0x0000_007f
	external_connection	Conduit	**led_wire**			
✓	⊟ **sdram**	SDRAM Controller Int...				
	clk	Clock Input	Double-click to	**sdram_pll_c0**		
	reset	Reset Input	Double			
	s1	Avalon Memory Mapped...	Double			_ffff
	wire	Conduit	**sdram_wire**			
✓	⊟ **sdram_pll**	ALTPLL Intel FPGA IP				
	inclk_interface	Clock Input	Double-click to	**clk_0**		
	inclk_interface_reset	Reset Input	Double-click to	[inclk_interface]		
	pll_slave	Avalon Memory Mapped...	Double-click to	[inclk_interface]	**0x0000_0080**	0x0000_008f
	c0	Clock Output	Double-click to	sdram_pll_c0		
	c1	Clock Output	**sdram_clk**	sdram_pll_c1		
✓	⊟ **sysid_qsys_0**	System ID Peripheral...				
	clk	Clock Input	Double-click to	**clk_0**		
	reset	Reset Input	Double-click to	[clk]		
	control_slave	Avalon Memory Mapped...	Double-click to	[clk]	**0x0000_0098**	0x0000_009f
✓	⊟ **switch**	PIO (Parallel I/O) I...				
	clk	Clock Input	Double-click to	**clk_0**		
	reset	Reset Input	Double-click to	[clk]		
	s1	Avalon Memory Mapped...	Double-click to	[clk]	**0x0000_0060**	0x0000_006f
	external_connection	Conduit	**switch**			
✓	⊟ **key3**	PIO (Parallel I/O) I...				
	clk	Clock Input	Double-click to	**clk_0**		
	reset	Reset Input	Double-click to	[clk]		
	s1	Avalon Memory Mapped...	Double-click to	[clk]	**0x0000_0050**	0x0000_005f

sdram
SDRAM Controller Intel FPGA IP [altera_avalon_new_sdram_controller 18.1]

Figure 3: General overview of the QSF file

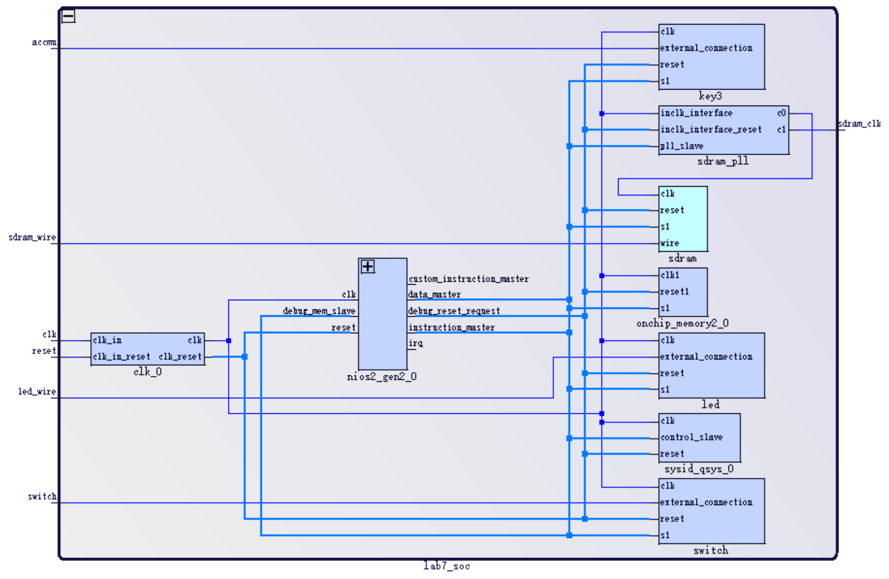


Figure 4: System Level Block Diagram

2.5 Answers to all 11 INQ Questions

What are the differences between the Nios II/e and Nios II/f CPUs? NIOS II/e is economy version of the processor which uses least resources and logic elements, it works more slowly than NIOS II/f, for it needs to take more clock cycle to complete one instruction.

What advantage might on-chip memory have for program execution? on-chip memory is located on the chip and designed to achieve data quickly at high frequency, we access off-chip memory more slowly because of longer data path.

Note the bus connections coming from the NIOS II; is it a Von Neumann, "pure Harvard", or "modified Harvard" machine and why? It is a modified Harvard machine. Von neumann machine has shared instruction and data memory and also shared instruction and data bus, pure Harvard machine has both two separated. For modified Harvard machine like NIOS II, the memory of instruction and data are also shared, but they have two separate path to transfer data.

Note that while the on-chip memory needs access to both the data and program bus, the led peripheral only needs access to the data bus. Why might this be the case? we only need led peripherals to get access to data path to display the data we have instead of instruction. On-chip memory need store both instruction and data.

Why does SDRAM require constant refreshing? Because SDRAM is made up of several capacitors and transistors, and they will decay with time. if we do not refresh it quite often, it can not remain correct when we access the data.

SDRAM parameter	Short name	Parameter value(fill in from datasheet)
Data Width	[width]	32 bits
# of Rows	[nrows]	13 bits
# of Columns	[ncols]	10 bits
# of Chip Selects	[ncs]	1 bits
# of Banks	[nbanks]	4 bits

Table 1: Parameters of SDRAM controller

Determine the following parameters to instantiate the SDRAM controller

What is the maximum theoretical transfer rate to the SDRAM according to the timings given? access time is 5.5ns, with 32 bits data transfer at one time. so the transfer rate should be $32bit/5.5ns = 727MB/s$

The SDRAM also cannot be run too slowly (below 50 MHz). Why might this be the case? There are capacitors in the SDRAM. If SDRAM runs too slowly without frequent refresh, its voltage will get decreased and its data value may not remain correct.

Make another output by clicking clk c1, and verify it has the same settings, except that the phase shift should be -3ns. This puts the clock going out to the SDRAM chip (clk c1) 3ns ahead of the controller clock (clk c0). Why do we need to do this? Hint, check Altera Embedded Peripheral IP datasheet under SDRAM controller we need to delay the clock of SDRAM for 3ns because SDRAM controller also needs time to access data and we have to wait for control signal to be correct at the address pin. Therefore, after 3ns, we can guarantee the control signal is valid and we are ready to address data.

What address does the NIOS II start execution from? Why do we do this step after assigning the addresses? NIOS II start exectution from the address x0200 0000 of SDRAM. We need to do this because we need go to our first instruction on that address and do not want to go to memory of data or anything irrelevant.

You must be able to explain what each line of this (very short) program does to your TA. Specifically, you must be able to explain what the volatile keyword does (line 8), and how the set and clear functions work by working out an example on paper (lines 13 and 16). Volatile keyword means the object can always be changed at anytime and represents hardware design and is written into from outside. We always use volatile to make pointers of hardware module. in code line 13 and 14, is SET function. we set i from 0 to 100000 as delays and make the least significant bit in LED OR 1 which must be 1 therefore we set the rigt most led light up. In line 15 and 16, it's clear function, we AND least signficant bit with x0001, which is always 0, therefore we successfully clear it.

segment	meaning	example
.bss	region of uninitialized parameter	int A;
.heap	dynamically allocated parameter.	int pointer = (int)malloc(sizeof(int));
.rodata	region of read only constant variables	const int A = 0;
.rwdata	region of data that can be changed	int A =0;
.stack	allocated variables and function calls	int func(int a, int b)
.text	region of string	char A = "hello";

Table 2: Design Resources and Statistics

Look at the various segment (.bss, .heap, .rodata, .rwdata, .stack, .text), what does each section mean? Give an example of C code which places data into each segment

3 Postlab Question

3.1 Document the Design Resources and Statistics in following table.

LUT	2247
DSP	0
Memory(BRAM)	36864 bits
Flip-Flop	1942
Frequency	65.49MHz
Static Power	98.54mW
Dynamic Power	7.9mW
Total Power	178.36mW

Table 3: Design Resources and Statistics

4 Conclusion

4.1 Discuss functionality of your design. If parts of your design didn't work, discuss what could be done to fix it

Our design works successfully. Our initial issue is the understanding of the relation of IP blocks with software and how they are connected. After building IP blocks in platform design and make their connections, we know better about the structure of hardware and software, NIOS-II and other hardware devices. We know that data and instruction will be stored in SDRAM and NIOS-II will do operations to read and write from SDRAM, and do some computations basing on the given data.

4.2 Was there anything ambiguous, incorrect, or unnecessarily difficult in the lab manual or given materials which can be improved for next semester? You can also specify what we did right so it doesn't get changed.

Most of part in lab manual is well. I think if you could add some overall block diagram of the whole structure and how software and hardware are related, it will be easier for us to understand. Besides, common errors like timestamp-error can be included more in the lecture or the lab manual.